Dealing with AADL end-to-end Flow Latency with UML Marte.

Su-Young Lee, Frédéric Mallet, Robert de Simone

UML&AADL'08. Belfast, Northern Ireland. April 2008, pp. 228-233
The original publication is available from http://dx.doi.org/10.1109/ICECCS.2008.14

Dealing with AADL End-to-end Flow Latency with UML MARTE

Su-Young Lee, Frédéric Mallet, Robert de Simone
Aoste project, INRIA Sophia Antipolis, FRANCE
{Su-Young.Lee, Frederic.Mallet, Robert.de_Simone}@sophia.inria.fr

Abstract

AADL and MARTE are both modeling formalisms supporting the analysis of real-time embedded systems. We investigate how MARTE, with its Time Model facilities, can be made to represent faithfully AADL periodic/aperiodic tasks communicating through event or data ports, in an approach to end-to-end flow latency analysis.

1. Introduction

Modern embedded systems must support the deployment of heterogeneous applications onto heterogeneous architectures. At design time the targeted execution platform may still be speculative, or the same applications may tentatively be deployed onto various flexible architectures. Therefore, early performance estimation of the pairing, under imposed real-time constraints, is highly desirable. Such analysis requires a model of both the application and the architecture, and effective means to define the mapping of applicative functions onto architecture resources and services.

AADL and MARTE are two such modeling frameworks. AADL (Architecture Analysis & Design Language) [1] was developed as a standard of the Society of Automotive Engineers (SAE), whereas MARTE (Modeling and Analysis of Real Time and Embedded systems) [2] is a recent OMG UML profile. Despite their many similar features (and MARTE being more detailed on the modeling aspects), AADL provides specific communication schemes between tasks, that need to be represented in MARTE: AADL tasks may be periodic or aperiodic, and in the former case of harmonic or independent periods; communication between tasks may use event-data or pure-data ports (with events triggering the recipient task behavior, while pure data are only sampled and used as such whenever the consuming tasks is otherwise activated).

Representing all these kinds of communications (periodic vs. aperiodic, event-triggered vs. sampled data) in MARTE is not only a challenge, but also an opportunity to provide timed semantics inside the modeling framework (and not aside, as separately provided semantic interpretation to time attributes). We build this semantic construction using MARTE Time Model [3], which is intended exactly for this: specifying in a formal way new timed domains of computation and communication.

We exemplify this approach by dealing with the same example as used in AADL [4] to explain the computation of end-to-end flow latencies (and various other related features) in a case of three threads with various rates (periodic or not) and connected through even-data or pure-data links in several locations. We show how these formulas are derived from time relations that are integral parts of the MARTE model.

2. A brief AADL overview

AADL supports the modeling of application software components (thread, subprogram, process), execution platform components (bus, memory, processor, device) and the ***binding*** of software onto execution platform. Each model element (software or execution platform) must be defined by a type and comes with at least one implementation.

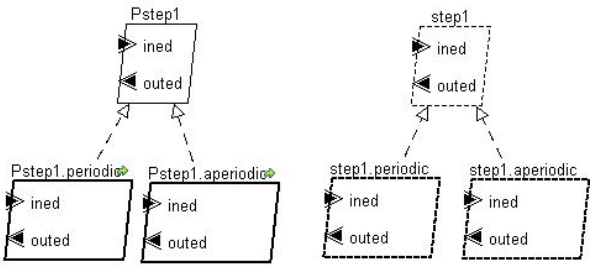

Figure 1. Declaration and implementation of both a process and a thread.

Figure 1 presents the declarations of one process type and one thread type. Each of these declarations

UML&AADL'08. Belfast, Northern Ireland. April 2008, pp. 228-233
The original publication is available from http://dx.doi.org/10.1109/ICECCS.2008.14

comes with the declaration of two possible implementations, one periodic and one aperiodic. Since threads are executed within the context of a process, the process implementations must specify the number of threads it executes and their interconnections. Figure 2 illustrates the case where a process executes two threads (t1 and t2) sequentially.

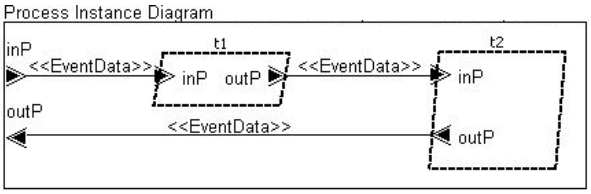

Figure 2. Two threads executed sequentially.

Type and implementation declarations also provide a set of properties to characterize model elements. For threads, the AADL standard properties include the dispatch protocol (periodic, aperiodic, sporadic, background), the period (if the dispatch protocol is periodic or sporadic), the deadline, the minimum and maximum execution times, along with many others.

AADL end-to-end flows explicitly identify a data-stream from sensors to the external environment (actuators). Figure 3 represents such a flow. Note that this model is really an excerpt of the complete model and implies that the threads are declared within the context of one or many processes and bound to an execution host, i.e., a processor. All this contextual information is absolutely required to make the latency analysis since the execution platform and the topology determines the actual parallelism available. When executing on a single processor platform, the threads have to be serialized whereas a dual processor platform offers more parallelism. Figure 8, detailed later, gives a more faithful view of the complete model.

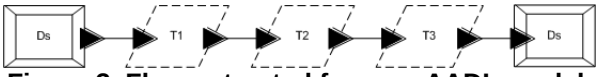

Figure 3. Flow extracted from an AADL model.

There are three kinds of ports: data, event and event-data. Data ports are for data transmissions without queuing. Connections between data ports are either immediate or delayed. Event ports are for communications of events that may be queued. The size of the queue may induce transfer delays that must be taken into account when performing latency analysis. Event data ports are for message transmission with queuing, here again the queue size may induce transfer delays. On Figure 3, all components have event-data ports represented as a solid triangle (as for data ports) with its shadow (as for event ports).

3. MARTE for AADL

The emerging UML2 Profile for MARTE is expected to be the basis for UML representation of AADL models [5]. The adopted MARTE OMG specification provides guidelines in this direction. The main goal of this paper is to further investigate how specific AADL concepts required for end-to-end flow latency analysis can be represented in MARTE. As such, this work is not (yet?) included in the official standard annex.

The idea here is to define, once and for all, a model library for AADL with MARTE. The end-user is not expected to enter into each and every detail about this library and most of the time he should be able to use it as a black box. The following section illustrates the use of this library on two selected examples. For brevity, we only present here the model elements required for dealing with the latency analysis example.

3.1. AADL application software components with MARTE.

The first step is to create classifiers to represent AADL threads. We use the stereotype SwSchedulableResource from the Software Resource Modeling sub-profile (see Figure 4). Only classifiers for *periodic* and *aperiodic* threads are shown here.

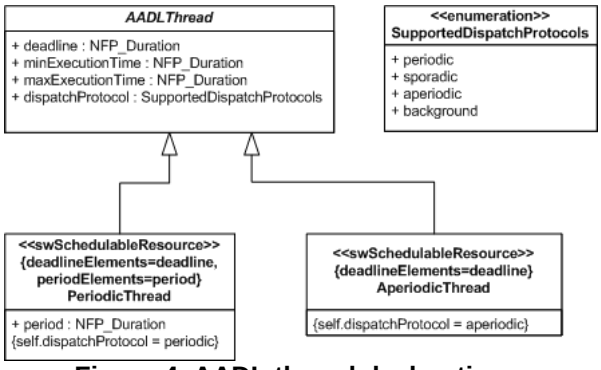

Figure 4. AADL thread declarations.

The properties deadlineElements and periodElements are explicitly identified so as to help model transformation tools to extract the right property. This makes it easier for the transformation tools to be language and domain independent. Hence, the exact spelling of the property names does not matter as long as they are referenced by the stereotype properties. Note that contrary to AADL only periodic threads have a property called period. The MARTE equivalent to the AADL type Time is NFP_Duration, defined in the MARTE::BasicNFP_Types (Non

UML&AADL'08. Belfast, Northern Ireland. April 2008, pp. 228-233
The original publication is available from http://dx.doi.org/10.1109/ICECCS.2008.14

Functional Property Types) model library. An NFP_Duration value is defined as a tuple containing a real value and a time unit, among others. Figure 5 shows examples of thread instances, the left one being periodic and the right one being aperiodic.

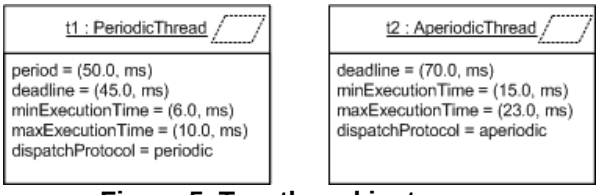

Figure 5. Two thread instances.

NFP_Duration only supports the description of duration values associated with an ideal chronometric clock, which is exactly what AADL supports. Had we wanted to support logical clocks we would have used the templateable TimedValueType defined in the MARTE::Time model library.

3.2. AADL hardware components with MARTE.

The Hardware Resource Modeling subprofile is used to model AADL hardware components (bus, memory, processor, and device). Possible equivalents using MARTE are given in Figure 6.

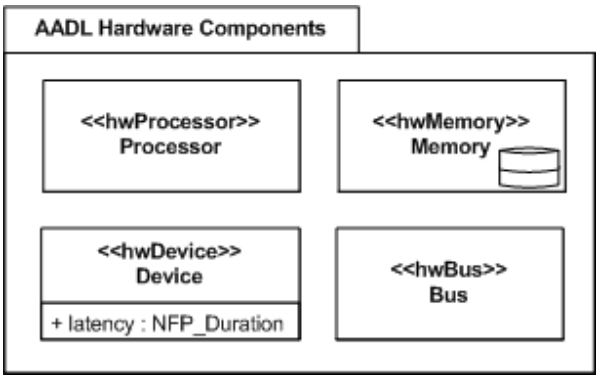

Figure 6. AADL hardware components.

Actually specific classifiers should be customized depending on the physical characteristics of the hardware components (throughput or latency of memories, clock speed of processors, etc.). The latency analysis performed here does not use any specific property, apart from the device latency, so these classifiers need not being specialized further.

3.3. AADL ports with MARTE.

UML component diagrams provide ports and connectors to connect components. The queuing policy should rather be represented on the algorithm itself, i.e., on a UML activity diagram. Activities are composed of actions. Ordering in which the actions are executed are given by a control flow. Data communications between the actions are represented with object flows. By default, an object flow has a queue, the size of which can be parameterized with its property upperBound. So object flows can be used to represent AADL communications using either event or event-data ports. UML allows the specification of a customized selection policy to select which one of the tokens stored in the object node is selected. Unfortunately, the selection behavior must select only one token making it impossible to represent the AADL dequeue protocol AllItems. This protocol dequeues all items from the port every time the port is read. Thus, only the dequeue protocol OneItem is supported.

To model data ports, UML provides DataStore nodes. On these nodes, the tokens are never consumed thus allowing for multiple readings of the same token. Using a data store node with an upper bound equal to one is a good way to represent communications through data ports.

On Figure 7, the upper part shows an event-based communication with a queue size of 4. The lower part illustrates a data-based communication.

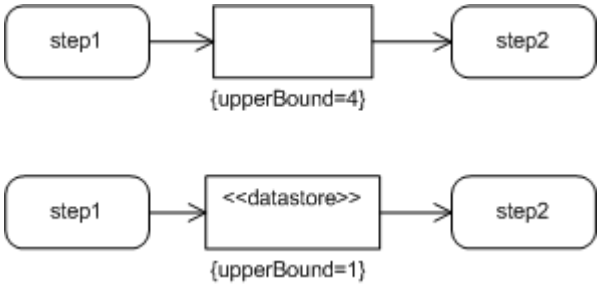

Figure 7. Event or data communications.

The difference between immediate and delayed communications is addressed in the next sub section, since it is not really a structural matter but rather a temporal aspect.

3.4. AADL MoCC with MARTE.

Aside the model elements, the time semantics of these elements must be defined. On one hand, the model of computation, i.e., when the processing starts, finishes or is aborted. On the other hand, the model of communications, i.e., what kind of communication is used. The MARTE Time subprofile, inspired from the theory of tags systems [6], provides a set of general mechanisms to define MoCC. These modeling aspects should be hidden to end-users and we show here how to use MARTE, as a model architect, to build a partial

UML&AADL'08. Belfast, Northern Ireland. April 2008, pp. 228-233
The original publication is available from http://dx.doi.org/10.1109/ICECCS.2008.14

MoCC suitable for AADL. To specify the time constraints with MARTE we use the stereotype ClockConstraint that extends the metaclass UML::Constraint. The language to be used on these clock constraints is called Clock Constraint Specification Language (CCSL) and is defined as an annex of the MARTE specification.

Overall two kinds of communications are possible in AADL. Data-driven communications, where the execution of a given task is triggered by the availability of the data produced by the preceding (according to the ordering defined by the control flow) task(s). Sampled communications, where pure data are only sampled and used as such whenever the task is otherwise activated. Note that the nature (event, event-data, or data) of the ports involved in the communication is not enough to determine its kind.

For instance, data-driven communications exist in chains of aperiodic tasks (devices or threads) connected by event or event-data ports. They also exist with synchronous periodic tasks connected by data-ports through an immediate connection. In that latter case, the consuming task becomes aperiodic and its execution is triggered by the completion of the producing task. The CCSL clock relation alternatesWith models data-driven communications.

step1.finish **alternatesWith** step2.start (1)

Eq. 1 illustrates a data-driven communication from step1 to step2. Note that this constraint is not symmetrical since the completion of step1 may cause the execution of step2, but not the converse.

Sampled communications occurs in various cases. For instance, when two asynchronous tasks (whether periodic or not) communicate, but also when two synchronous periodic tasks are connected by data-ports through a delayed connection.

step2.start = step1.finish **sampledTo** ^step2 (2)

Eq. 2 illustrates a sampled communication from step1 to step2. ^step2 represents the activation condition of step2. If step2 is a periodic thread, its activation condition can be defined using the CCSL relation isPeriodicOn (see Eq. 3–4).

c_{100} ≡ idealClk **discretizedBy** 0.01 (3)

^step2 **isPeriodicOn** c_{100} period=10 (4)

idealClk is defined in the MARTE Time library and stands for a dense chronometric (related to physical time) perfect (with no jitter or any other flaw) clock. Eq. 3 defines c_{100} by discretizing idealClk. The default unit of idealClk is the second (s), so c_{100} is a 100-hz discrete chronometric clock. ^step2 is periodic on c_{100} with period 10 (Eq. 4), that makes ^step2 a 10-hz discrete chronometric clock.

Had step2 been an aperiodic thread, its activation condition would have been an unbound logical clock.

4. An example

4.1. The AADL description

In this section we combine all these elements into a complete model that derives from [4]. It is displayed using the OSATE Eclipse plug-in environment for AADL [7] in Figure 8.

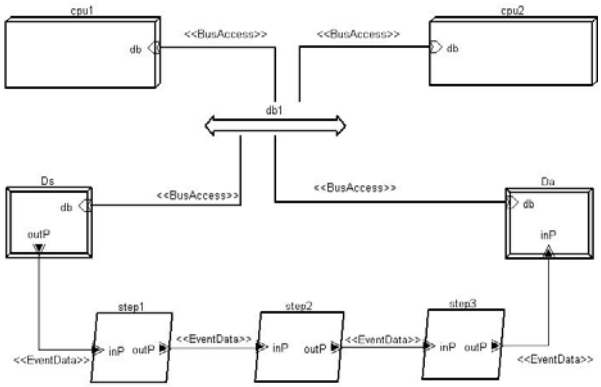

Figure 8. The System in OSATE

This flow starts from a sensor (an aperiodic device instance) and sinks in an actuator (also aperiodic) through three process instances. Each process executes a single thread. The two devices are part of the execution platform and communicate via a bus (db1) with two processors (cpu1 and cpu2), which host the three processes with several possible bindings. All processes are executed by either the same processor, or any other combination. The actual binding is not represented on this figure as we have ignored the effects of communications, just as in the original example [4]. The component declarations and implementations are not presented here. The full AADL code is available in [4].

4.2. The MARTE representation (synchronous sampling flow case)

We start by describing the model algorithm with an UML activity diagram (see Figure 9, upper-most part). All communications are through event-data ports with infinite queues. Two actions (acquire and release) have been added as the behavior of devices, compare with Figure 3.

UML&AADL'08. Belfast, Northern Ireland. April 2008, pp. 228-233
The original publication is available from http://dx.doi.org/10.1109/ICECCS.2008.14

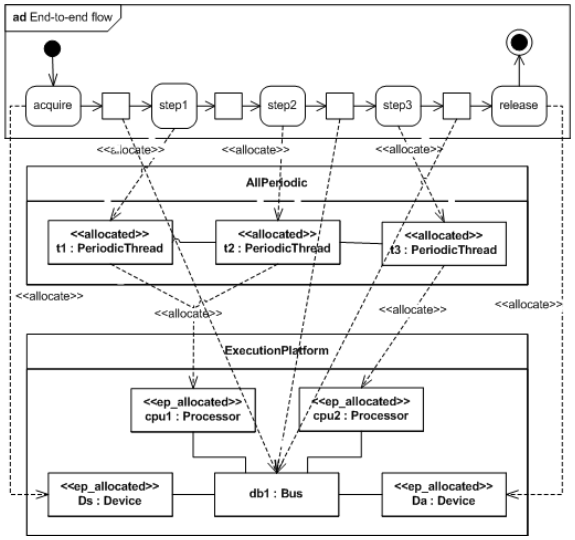

Figure 9. MARTE model, all periodic case.

AADL software (Figure 9, middle part) and execution platform (Figure 9, lower part) components are modeled with composite structure diagrams using the classifiers defined in Section 3. The information extracted from the MARTE stereotypes helps defining adequate clock constraints (Figure 10) following the methodology defined in Section 3.4.

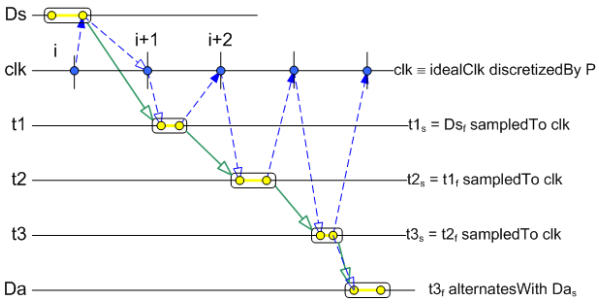

Figure 10. CCSL constraints, all-periodic case.

Since all threads are periodic and synchronous, the first three communications (from acquire to step1, from step1 to step2, and from step2 to step3) are sampled communications. The last communication (from step3 to release) is data-driven, since the device is aperiodic.

$$clk \equiv idealClk \textbf{ discretizedBy } P \quad (5)$$

Eq. 5 declares the common clock to all periodic threads. Figure 10 represents the CCSL constraints and an equivalent graphical representation. Green arrows represent the communications. Dashed arrows represent the instant relations. Plain arrows denote strict precedence whereas empty arrows denote non-strict precedence. The first two lines can then be interpreted as follows. The instant at which the action acquire (Ds) finishes (Ds.finish) is located between two ticks of the clock, let clk_i be this instant (Eq. 6).

$$(\exists i \in \mathbb{N}^*)\ (Ds.finish \in]clk_i, clk_{i+1}]) \quad (6)$$

This implies that the action step1 must follow the tick clk_{i+1} (Eq. 7).

$$step1.start \in [clk_{i+1}, clk_{i+2}[\quad (7)$$

That is the definition of an asynchronous sampling. The data emitted by the asynchronous device Ds is sampled by the synchronous thread t1, according to its clock clk.

The AADL binding mechanism finds its equivalent in the MARTE allocation package. First, actions and object nodes are allocated (dashed arrows on Figure 9) to software components. Second, software components are allocated to execution platform model elements.

All these annotations (stereotypes) can be extracted using model-driven engineering techniques and fed into time analysis tools, including AADL latency analysis tool. Then, we go a bit further than AADL, by bringing back the latency analysis results into UML and MARTE in the form of timing diagrams (Figure 11). The timing diagram represents a family of possible schedules for a given execution flow and a given pair application/execution platform.

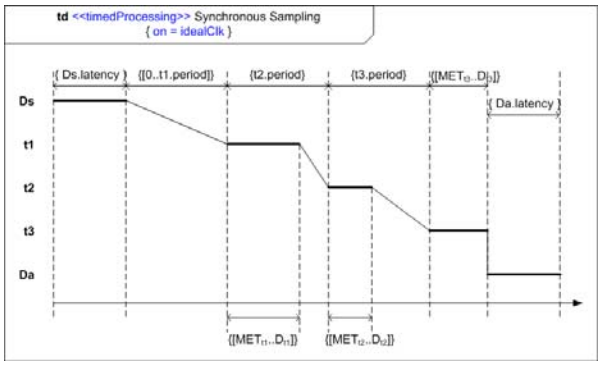

Figure 11. Timing diagrams, all periodic case.

Computation execution times (bold horizontal lines) are equal to the latency for devices and range between the MinimumExecutionTime and the Deadline for threads. Oblique lines linking two computation lines represent the communications and the sampling delays. For sampled communications, this amounts to wait for the next tick of the receiver clock. The maximal sampling delay is when the communication waits for the full sampling period because the previous tick has just been missed. It is not normative in UML

UML&AADL'08. Belfast, Northern Ireland. April 2008, pp. 228-233
The original publication is available from http://dx.doi.org/10.1109/ICECCS.2008.14

timing diagrams to have these "oblique" lines, but it is a convenient notation to represent intermediate *communication* states between two steady *processing* states (e.g., between Ds and t1). Assuming, as in [4], that the sampling delays are always maximal, we get the same formulas (reproduced below) as the AADL latency analysis tool.

End-to-End Flow Latency = Ds.latency + flow latency + Da.latency

$\text{Flow Latency}_{\text{worst-case}}$ = t1.period + t2.period + t3.period + t3.deadline

$\text{Flow Latency}_{\text{Best-Case}}$ = t1.period + t2.period + t3.period + t3.MinExecTime

Latency jitter = t3.deadline – t3.MinExecTime

4.3. The MARTE representation (Mixed Event-data flow case)

We study here a second possible configuration extracted from [4] that only differs by making aperiodic the thread t2 (Figure 12). Few other cases involving data ports are studied in [8].

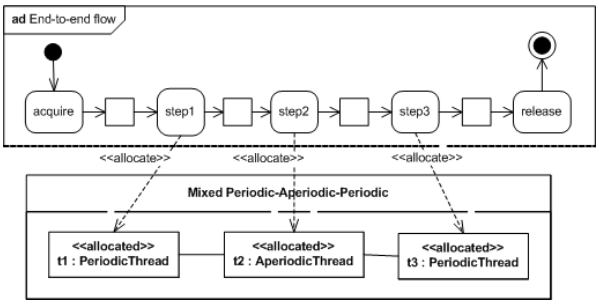

Figure 12. MARTE model, mixed case.

In that configuration, the communication from step1 to step2 becomes data-driven. The CCSL constraint is adapted accordingly (Figure 13). We also get a different timing diagram (Figure 14) and different flow latency formulas.

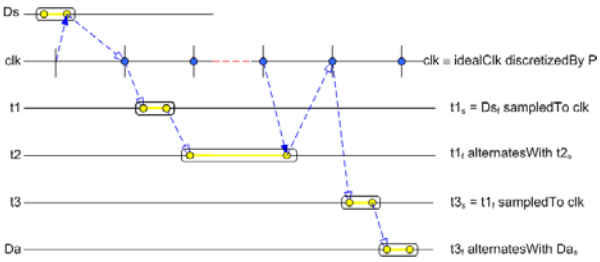

Figure 13. CCSL constraints, mixed case.

End-to-End Flow Latency = Ds.latency + flow latency + Da.latency

$\text{Flow latency}_{\text{Worst-Case}}$ = t1.period + k1 * t3.period + t3.deadline

$\text{Flow Latency}_{\text{Best-Case}}$ = t1.period + k2 * t3.period + t3.MinExecTime (k2≤k1)

Latency jitter = t3.deadline – t3.MinExecTime + (k1-k2)*t3.period

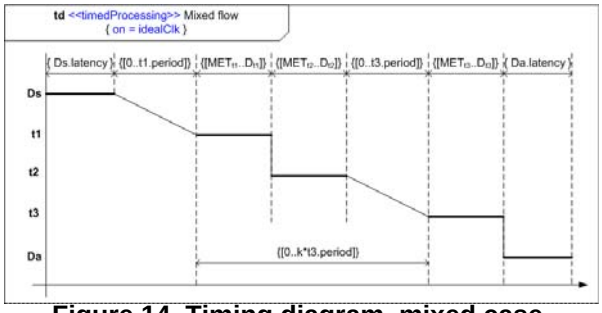

Figure 14. Timing diagram, mixed case.

5. Conclusion

We have shown how MARTE could be used to model mixed systems with both periodic and aperiodic tasks, which is a big issue while modeling embedded systems. We have compared MARTE and AADL in this matter, highlighting MARTE capabilities to make the computation formulas explicit. Two different configurations are illustrated.

More generally, MARTE and its time model could be used to model various timed models of computation and communication.

6. References

[1] SAE: Architecture Analysis and Design Language (AADL). June 2006, document AS5506/1. http://www.sae.org/technical/standards/AS5506/1.

[2] OMG: UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), beta 1, August 2007, document ptc/07-08-04.

[3] André C, Mallet F, de Simone R (2007): Modeling Time(s). Springer LNCS 4735:559-573.

[4] Feiler P.H, Hansson J: Flow Latency Analysis with the Architecture Analysis and Design Language (AADL). Carnegie Mellon University, Technical Note CMU/SEI-2007-TN-010, June 2007.

[5] Faugère M, Bourbeau T, de Simone R, Gérard S (2007): MARTE: Also an UML Profile for Modeling AADL Applications. ICECCS:359-364.

[6] Lee E.A, Sangiovanni-Vincentelli A.L (1998): A framework for comparing models of computation. IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems, 17(12):1217-1229.

[7] OSATE release 1.5.3. http://www.aadl.info

[8] André C, Mallet F, de Simone R (2007): Modeling of Immediate vs. Delayed Data Communications: from AADL to UML MARTE". ECSI FDL 2007.